化工应用数学 第四章 方程(组)数值求解 讲义

01. 线性方程组

课程回顾及后续内容：

本课程之前主要讲述了以下两方面的内容：

1. Python 语言及编程
   - 语法：基础语法、判断语句、循环语句、函数
   - 库：SciPy 实际上是很多 Python 库的集合，包括 Sympy、pandas、NumPy、Matplotlib 等
2. 数据处理
   - 插值算法：拉格朗日插值
   - 数值微分：差商法、插值法
   - 数值积分：牛顿-科斯特公式
   - 拟合算法：最小二乘法、最小二乘拟合

在之后的课程中，我们将主要讲解以下四个部分：

- 方程(组)数值求解
- 常微分方程数值求解
- 偏微分方程数值求解
- 人工智能-神经网络

而对于傅里叶/拉普拉斯变换、场论、概率论与数理统计、数据校正和图论等，因为时间的关系不在本门课程讲述范围之内，但下面会简单给大家介绍一下相关的内容。

傅里叶变换简单来说就是把函数表示为三角函数加和的形式，这一点其实在现实生活中我们经常能够见到，就是彩虹——白光会被分解为不同颜色的光，这里就表示白光是可以分解为不同频率(颜色)的光的，这里我们不就具体内容详细讲解，但是大家可以记住最后的结论：

当满足狄利赫里条件时，傅里叶变换及其逆变换为：

$$F(\omega)=\int_{-\infty}^{+\infty}f(t)e^{-i\omega t}dt$$

$$f(t)=\frac{1}{2\pi}\int_{-\infty}^{+\infty}F(\omega)e^{i\omega t}d\omega$$

下面这个例子给大家傅里叶变换能力的概念，对于任意图形，我们都能够使用傅里叶变换来复现这个图形。思考：插值算法是否能够实现这一过程？

傅里叶变换可以看作是拉普拉斯变换的一个特例，我们可以简单的理解拉普拉斯变换主要是为了解决函数不满足狄利赫里条件的情况，通过乘上一个因子使得函数可积，同时只考虑正半轴：

$$F(\omega)=\int_0^{+\infty} f(t)e^{-\sigma t}e^{-i\omega t}dt=\int_0^{+\infty} f(t)e^{-(\sigma+i\omega)t}dt$$

我们把括号里的式子记为 s，则这个公式就是拉普拉斯变换。

场论则主要是通过矩阵/数组的形式表示特定物理量在空间上的分布，通过数学公式的形式表达这些物理量的变化，这里涉及到的知识除了上面说的那些控制方程之外，还主要包括了标量与向量的操作运算：梯度、散度、旋度。

数学建模：

数学建模就是根据实际问题来建立数学模型，对数学模型来进行求解，然后根据结果去解决实际问题。

当需要从定量的角度分析和研究一个实际问题时，人们就要在深入调查研究、了解对象信息、作出简化假设、分析内在规律等工作的基础上，用数学的符号和语言作表述来建立数学模型。

根据我们所建立的数学模型和所要求解的对象不同，求解过程中就可能会分别需要后面所讲的内容，或者可以说应用数学本身就是在建模的基础之上再对实际问题进行处理和解决。

线性方程求解：（回顾线性代数知识）

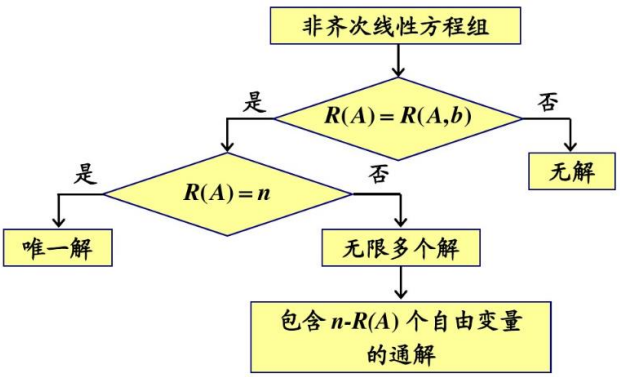
$$\begin{cases} a_{11}x_1+a_{12}x_2+\cdots+a_{1n}x_n=b_1, \\ a_{21}x_1+a_{22}x_2+\cdots+a_{2n}x_n=b_2, \\ \cdots\cdots\cdots\cdots \\ a_{m1}x_1+a_{m2}x_2+\cdots+a_{mn}x_n=b_m \end{cases}$$

线性方程组的一般形式如上图，一般的我们简记为：Ax=b

根据线性代数的知识，我们知道，这一方程组的解的情况如下：

- 如果 R(A)<R(A,b)：无解
- 如果 R(A)=R(A,b)=n：有唯一解
- 如果 R(A)=R(A,b)<n：有无穷多解

所以，一般的在求解线性方程组的时候，我们会参照下面的流程：



下面以例题的形式来具体回顾一下线性方程组的求解过程：

例2　求解非齐次方程组的通解

$$\begin{cases} x_1 - x_2 - x_3 + x_4 = 0 \\ x_1 - x_2 + x_3 - 3x_4 = 1 \\ x_1 - x_2 - 2x_3 + 3x_4 = -1/2 \end{cases}.$$

解　对增广矩阵B进行初等变换

$$B = \left(\begin{array}{cccc|c} 1 & -1 & -1 & 1 & 0 \\ 1 & -1 & 1 & -3 & 1 \\ 1 & -1 & -2 & 3 & -1/2 \end{array}\right) \begin{matrix} \\ r_2-r_1 \\ r_3-r_1 \end{matrix} \sim \left(\begin{array}{cccc|c} 1 & -1 & -1 & 1 & 0 \\ 0 & 0 & 2 & -4 & 1 \\ 0 & 0 & -1 & 2 & -1/2 \end{array}\right)$$

$$\begin{matrix} r_2-r_1 \\ \sim \\ r_3-r_1 \end{matrix} \left(\begin{array}{cccc|c} 1 & -1 & -1 & 1 & 0 \\ 0 & 0 & 2 & -4 & 1 \\ 0 & 0 & -1 & 2 & -1/2 \end{array}\right)$$

$$\begin{matrix} r_2 \leftrightarrow r_2 \\ \sim \end{matrix} \left(\begin{array}{cccc|c} 1 & -1 & 0 & -1 & 1/2 \\ 0 & 0 & 1 & -2 & 1/2 \\ 0 & 0 & 2 & -4 & 1 \end{array}\right) \begin{matrix} \\ r_3-2r_2 \\ \sim \end{matrix} \left(\begin{array}{cccc|c} 1 & -1 & 0 & -1 & 1/2 \\ 0 & 0 & 1 & -2 & 1/2 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

由于$R(A) = R(B) = 2$, 故方程组有解，且有

$$\begin{cases} x_1 = x_2 + x_4 + 1/2 \\ x_3 = 2x_4 + 1/2 \end{cases} \Leftrightarrow \begin{cases} x_1 = x_2 + x_4 + 1/2 \\ x_2 = x_2 + 0x_4 \\ x_3 = 0x_2 + 2x_4 + 1/2 \\ x_4 = 0x_2 + x_4 \end{cases}$$

所以，方程组的通解为：

$$\mathrm{x} = c_1 \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} + c_2 \begin{pmatrix} 1 \\ 0 \\ 2 \\ 1 \end{pmatrix} + \begin{pmatrix} 1/2 \\ 0 \\ 1/2 \\ 0 \end{pmatrix}$$

以上是对于线性代数中线性方程组的回顾，下面我们主要讨论 m=n 的情况，这也是我们在实际中经常遇到的情况，并且在实际情况中我们所要处理的问题都是具有唯一解的情况，所以我们将主要处理这种情况。

对于这种情况，一般来说我们有两种方法：使用逆矩阵和使用克莱姆法则。

逆矩阵法顾名思义，方程 Ax=b 的解为 $x=A^{-1}b$，这一方法的难点在于如何求取系数矩阵的逆矩阵；而另一种方式是使用克莱姆法则，即：

$$x_j = \frac{D_j}{D}\ (j = 1, 2, \cdots, n)$$

其中 D_j 是把 D 中第 j 列元素对应地换成常数项所得到的行列式，这种方式的难度在于计算量太大。

克莱姆法则计算量：以上计算要计算 n+1 个行列式，每个行列式有 n！项，而每一项需要计算 n 个数的乘积，所以总的计算量为(n+1)n!(n-1)次乘积运算。

高斯消去法：

高斯消去法是一种常用的求解线性方程组的方法，通过逐次消元后，再回代求解；进行变换的目的是将系数矩阵按照从上至下、从左至右的顺序化为上三角矩阵，从而就能从下至上的逐步求解方程组；具体过程如下：

$$\boldsymbol{B}^{(1)} \Rightarrow \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} & b_n^{(2)} \end{pmatrix} \triangleq \boldsymbol{B}^{(2)}$$

$$\boldsymbol{B}^{(2)} \Rightarrow \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} & b_3^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & a_{n3}^{(3)} & \cdots & a_{nn}^{(3)} & b_n^{(3)} \end{pmatrix} \triangleq \boldsymbol{B}^{(3)}$$

最终需要转换为如下形式：

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(n)} \end{bmatrix}$$

消去过程计算量：$\sum_{k=1}^{n-1}(n-k+1)^2 = \frac{n(n+1)(2n+1)}{6} - 1$

回带过程计算量：$\sum_{i=n}^{1}(n-i+1) = \frac{n(n+1)}{2}$

所以高斯消去法总的计算量为：$\frac{n^3}{3} + n^2 + \frac{2n}{3} - 1 \approx \frac{n^3}{3} = O(n^3)$

高斯消去法原理清晰、编程简单，但是存在以下两方面限制：

（1）每次运算时，必须保证对角线上的元素不为 0（即运算中的分母不为 0），否则算法无法继续进行；

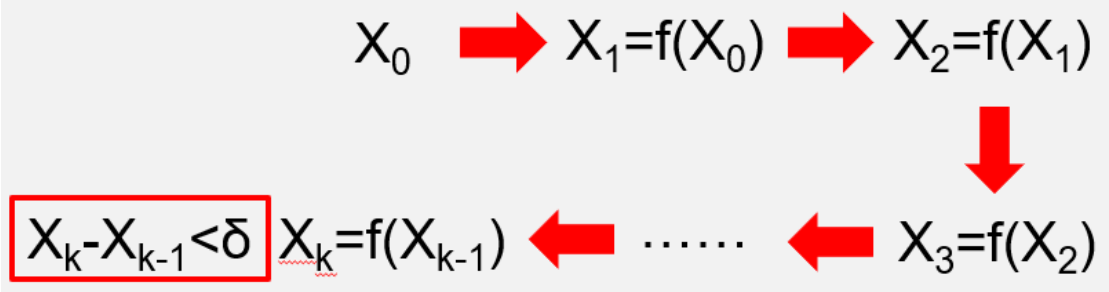
（2）即使主元素的值不为零，但如果绝对值很小，由于计算中主元素值在分母位置，因此作除数会引起很大的误差，从而影响算法的稳定性。

所以提出了相对应的修正方法：高斯主元素消去法，即选取绝对值尽可能大的元素作为主元素。

线性方程组的直接数值解法基本都是类似于高斯消元法，对这部分我们不再具体讲解，大家可以参照教材了解相关内容，如高斯-约当消去法、矩阵求逆矩阵、LU 分解等方法。

线性方程组迭代解法：

迭代方法的思想是利用所要求解的方程组，构造某种迭代格式，然后从某个猜测的解出发，逐步迭代，在某些特定情况下计算结果就会收敛于真解，其一般流程如下：



雅可比迭代法：将系数矩阵分解为三部分，A=D-L-U（如下图），那么原线性方程组转化为(D-L-U)X=b，从而 DX=(L+U)X+b，在此基础之上，我们构建雅可比迭代公式为：$X^{k+1}=D^{-1}(L+U)X^k+D^{-1}b$。

$$= \begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn} \end{bmatrix} - \begin{bmatrix} 0 & & & & \\ -a_{21} & 0 & & & \\ -a_{31} & -a_{32} & 0 & & \\ \vdots & \vdots & \vdots & \ddots & \\ -a_{n1} & -a_{n2} & \cdots & -a_{n,n-1} & 0 \end{bmatrix}$$

$$- \begin{bmatrix} 0 & -a_{12} & -a_{13} & \cdots & -a_{1n} \\ & 0 & -a_{23} & \cdots & -a_{2n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & -a_{n-1,n} \\ & & & & 0 \end{bmatrix}$$

高斯-赛德尔迭代法：在实际计算过程中，我们往往会发现雅可比迭代特别难以收敛，所以研究者们提出了一种使用同一迭代步长中已经计算出数值的方法（即在第 k 次迭代过程中，在计算 x_i 时，考虑了 x_1、x_2、…、x_{i-1}），体现在具体的计算过程中，使用的迭代公式为：$X^{k+1}=D^{-1}LX^{k+1}+ D^{-1}UX^k +D^{-1}b$

此外，类似的还有松弛迭代法，其引入松弛因子 ω，通过 ω 控制迭代的收敛速度等，相关具体内容可参考教材。

线性方程组的 Python 求解：

在 Python 中有很多库都提供了线性方程组的求解，比如我们之前讲过的 sciPy，这里为简单起见我们仅介绍最简单的使用 numpy 进行求解的方法。

Numpy 逆矩阵：Numpy 中求矩阵 A 的逆矩阵有两种方式，分为是 A.I 和 numpy.linalg.inv(A)，有了系数矩阵的逆矩阵之后使用其乘以常数向量即可得到方程组的解。

直接求解：对线性方程组 AX=b，可以使用 numpy.linalg.solve(A,b)直接求解。

下面以一个具体的例子来展示：

```
import numpy as np
a=np.mat([[1,2],[3,4]])
ia1=a.I
ia2=np.linalg.inv(a)
print('Coef Matrix:\n',a)
print('Inverse Matrix 1:\n',ia1)
print('Inverse Matrix 2:\n',ia2)
b=np.array([5,6])
x1=np.dot(ia1,b)
x2=np.dot(ia2,b)
x=np.linalg.solve(a,b)
print('solution with inv 1:\n',x1)
print('solution with inv 2:\n',x2)
print('solution with linalg:\n',x)
```

其结果显示为：

```
Coef Matrix:
 [[1 2]
 [3 4]]
Inverse Matrix 1:
 [[-2.   1. ]
 [ 1.5 -0.5]]
Inverse Matrix 2:
 [[-2.   1. ]
 [ 1.5 -0.5]]
solution with inv 1:
 [[-4.   4.5]]
solution with inv 2:
 [[-4.   4.5]]
solution with linalg:
 [-4.   4.5]
```